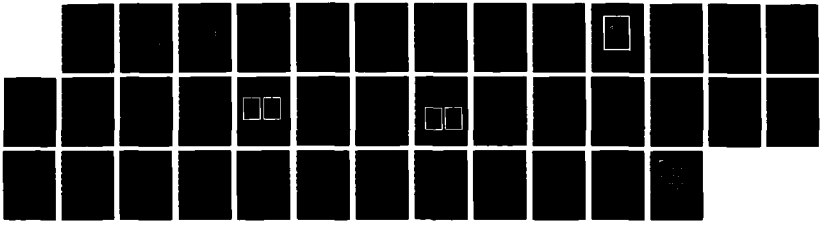
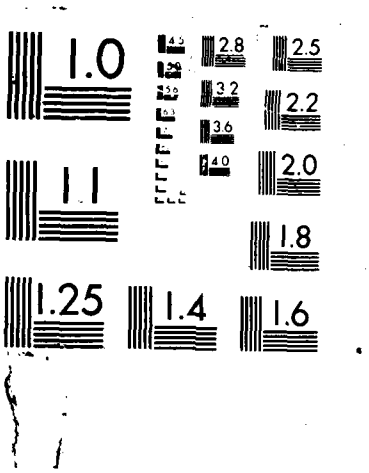
AD-A189 246 ROVER: A PROTOTYPE ACTIVE VISION SYSTEM(U) ROCHESTER UNIV NY DEPT OF COMPUTER SCIENCE D J COOMBS ET AL. AUG 87 TR-219 DACA76-85-C-0001 1/1

UNCLASSIFIED F/G 23/3 NL




1.0
1.1
1.25
1.4
1.6
1.8
2.0
2.2
2.5
2.8
3.2
3.6
4.0

ROVER: A Prototype Active Vision System

David J. Coombs Brian D. Marsh

The University of Rochester
Computer Science Department
Rochester, New York 14627

Technical Report 219

August 1987

ROVER: A Prototype Active Vision System

David J. Coombs Brian D. Marsh

The University of Rochester
Computer Science Department
Rochester, New York 14627

Technical Report 219

August 1987

Abstract

The Roving Eyes project is an experiment in active vision. We present the design and implementation of a prototype that tracks colored balls in images from an on-line CCD camera. **Rover** is designed to keep up with its rapidly changing environment by handling best and average case conditions and ignoring the worst case. This strategy is predicated on the assumption that worst case conditions will not persist for long periods of time and the system's limited resources should be directed at the problems which are likely to yield the most results for the least effort. This allows Rover's techniques to be less sophisticated and consequently faster. Each of Rover's major functional units is relatively isolated from the others, and an executive which knows all the functional units directs the computation by deciding which jobs would be most effective to run. This organization is realized with a priority queue of jobs and their arguments. Rover's structure not only allows it to adapt its strategy to the environment, but also makes the system extensible. A capability can be added to the system by adding a functional module with a well-defined interface and by modifying the executive to make use of the new module. The current implementation is discussed in the appendices.

This work was supported in part by NSF research grant number DCR-8602958 and in part by U.S. Army Engineering Topographic Laboratories research contract number DACA76-85-C-0001. We thank the Xerox Corporation University Grants Program for providing equipment used in the preparation of this paper.

SECURITY CLASSIFICATION OF THIS PAGE (When Data Entered)

REPORT DOCUMENTATION PAGE

READ INSTRUCTIONS BEFORE COMPLETING FORM

1. REPORT NUMBER: TR 219	2. GOVT ACCESSION NO.
3. RECIPIENT'S CATALOG NUMBER	
4. TITLE (and Subtitle): ROVER: A Prototype Active Vision System	5. TYPE OF REPORT & PERIOD COVERED: technical report
	6. PERFORMING ORG. REPORT NUMBER
7. AUTHOR(s): D.J. Coombs and B.D. Marsh	8. CONTRACT OR GRANT NUMBER(s): DACA76-85-C-0001
9. PERFORMING ORGANIZATION NAME AND ADDRESS: Dept. of Computer Science 535 Computer Studies Building Univ. of Rochester, Rochester, NY 14627	10. PROGRAM ELEMENT, PROJECT, TASK AREA & WORK UNIT NUMBERS
11. CONTROLLING OFFICE NAME AND ADDRESS: DARPA 1400 Wilson Blvd. Arlington, VA 22209	12. REPORT DATE: August 1987 13. NUMBER OF PAGES: 30
14. MONITORING AGENCY NAME & ADDRESS (if different from Controlling Office): Office of Naval Research Information Systems Arlington, VA 22217	15. SECURITY CLASS. (of this report): unclassified 15a. DECLASSIFICATION/DOWNGRADING SCHEDULE

16. DISTRIBUTION STATEMENT (of this Report)

Distribution of this document is unlimited.

17. DISTRIBUTION STATEMENT (of the abstract entered in Block 20, if different from Report)

18. SUPPLEMENTARY NOTES

19. KEY WORDS (Continue on reverse side if necessary and identify by block number)

active vision
dynamic scene
adaptive behavior

20. ABSTRACT (Continue on reverse side if necessary and identify by block number)

The Roving Eyes project is an experiment in active vision. We present the design and implementation of a prototype that tracks colored balls in images from an on-line CCD camera. Rover is designed to keep up with its rapidly changing environment by handling best and average case conditions and ignoring the worst case. This allows Rover's techniques to be less sophisticated and consequently faster. Each of Rover's major functional units is relatively isolated from the others, and an executive which knows all the functional

DD FORM 1 JAN 73 1473 EDITION OF 1 NOV 65 IS OBSOLETE

SECURITY CLASSIFICATION OF THIS PAGE (When Data Entered)

20. ABSTRACT (Continued)

units directs the computation by deciding which jobs would be most effective to run. This organization is realized with a priority queue of jobs and their arguments. Rover's structure not only allows it to adapt its strategy to the environment, but also makes the system extensible. A capability can be added to the system by adding a functional module with a well-defined interface and by modifying the executive to make use of the new module. The current implementation is discussed in the appendices.

Contents

List of Figures

1 Introduction

As part of an ongoing research program in active vision at University of Rochester's Computer Science Department [Ballard *et al.* 1987,Ballard 1987], the Roving Eyes project (Rover) is an experiment in the design and implementation of an active vision system. For this project, the task is the identification and tracking of moving (as well as stationary) objects. Images are taken by a fixed camera and are first analyzed to detect regions of light-colored blobs in a dark field. These areas are further analyzed to detect the specific identity of the objects in the scene. The results of this identification process are then incorporated into a database which represents the system's model of the world. With such a model, the system is capable of maintaining accurate correspondence between distinct objects over time. Hence, Rover represents a vision system with cognitive as well as sensory abilities.

As an example of Rover performing this sort of task, Figure 1 shows Rover's graphics display as it tracks the position of one ball from the lower left corner of the scene to the right and upward while another ball simultaneously moves from the upper right to the left.

The initial Rover prototype has been designed only to deal with distinctly colored spherical blobs (*e.g.*, balls, eggs, *etc.*). This simplification in the sensory task of object recognition allows us to address simple cognitive issues as well as sensory ones, thereby increasing the scope of the system. A descendent of Rover could collect enough information to perform such tasks as identifying a solitary block in a field of spheres or identifying particular blocks not just by color (which in fact is just the simple calculation of a moment) but by using alphabet blocks which may be distinguished by the letters on their faces.

2 Maintaining Correspondence—Strategic Issues

One of the major problems in tracking moving objects is the *correspondence problem*. Specifically, given two images, we want to be able to identify those regions in both images that represent the same object in the scene. The correspondence problem raises issues of cognition as well as sensation and is not easily solved by simple template matching techniques. An active vision system must cope with the trade-off between input bandwidth and the time required to perform the necessary perceptive and cognitive tasks. Evolution has balanced these trade-offs for animals fairly well; Rover must address the same problem, as it is a serial, cognition-limited system that must be flexible and adaptable. Keeping the sampling time interval short, however, requires Rover's cognitive analysis to be fairly fast, and consequently it must be simple. Due to the simplicity of these algorithms, we can only expect them to succeed most of the time, but we believe occasional failures are tolerable because Rover's world is constantly changing and will likely present more favorable data within a short time.

Rover's strategy for maintaining correspondence between objects in a scene is based on a separation between cognition (and attention) and sensation. Maintaining correspondence over time is a cognitive ability. In Rover, motion detection and object identification are lower-level, sensory problems. Sensory techniques (*e.g.*, for motion and blob detection) are used to analyze the current image and translate it into symbols that may be effectively ma-

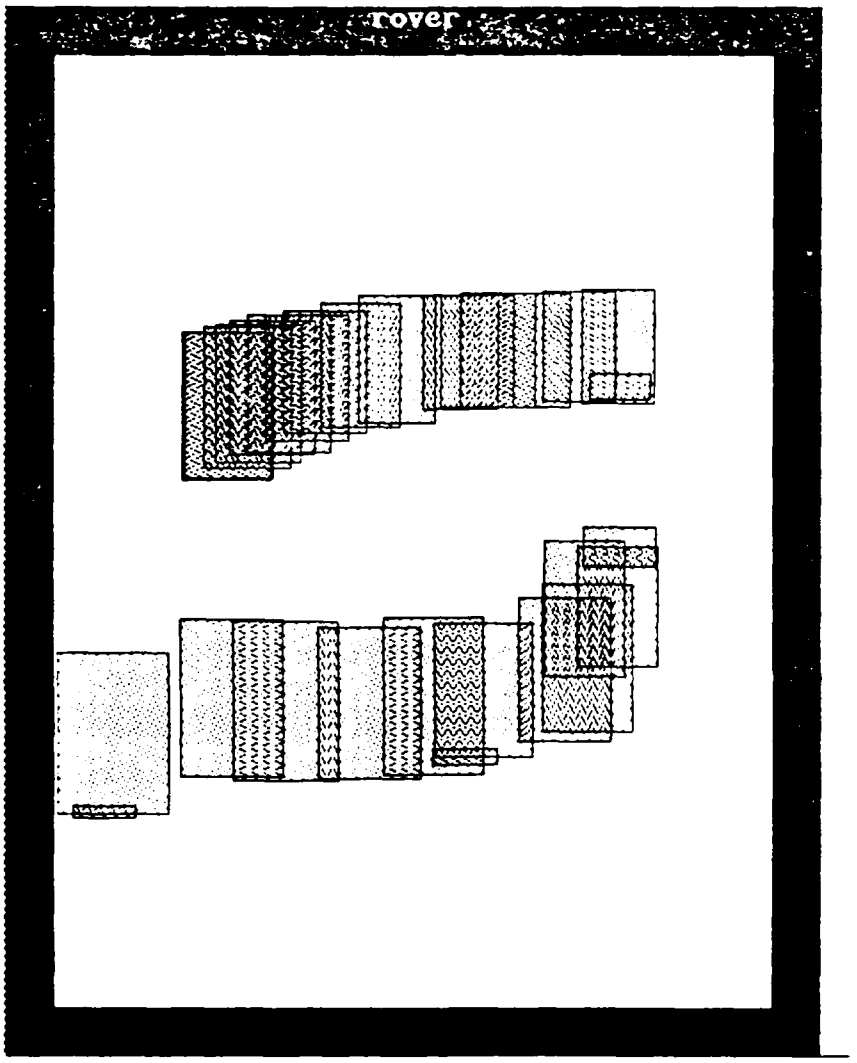


Figure 1: Graphics display of Rover tracking two balls simultaneously: the lower ball moves left to right, and the upper from right to left. During each frame interval, Rover locates the light-colored balls in the field and denotes each ball with a gray rectangle. Over time, the rectangles drift across the display (possibly overlapping each other) as the balls move in the field. Much smaller boxes appear occasionally; this occurs when the system mistakenly thinks a ball is smaller than it really is. Rover is robust against these problems.

nipulated by cognitive processes. These processes use this information not only to maintain correspondence but also to focus attention on areas of relative importance.

The most natural way to coordinate sensory tasks and cognitive tasks would be in an event driven semantic network. Such a structure run in parallel would present an extremely powerful organization. Unfortunately, the machine used for the implementation of the prototype is not a parallel architecture. To use the machine at our disposal most efficiently, we have devised a control structure that allows for an extremely fine grain multiplexing of resources. Sub-problems are kept small and are executed only when their relative importance is great enough to the overall functioning of the system. These sub-tasks are:

1. Several kinds of independent sensory analyses continuously process raw image data.
2. Cognitive processes focus (the more expensive) attentive resources (task type 3) on interesting parts of the scene. When they need input from the real world they sample the results from task 1.
3. There may be several simultaneous loci of attention and several attentive processes working on each locus at the same time.

It is worth noting that this organization is not inconsistent with parallel implementation. In particular, the independent analyses of task 1 and simultaneous attention to different aspects of the image in task 3 would be natural candidates for implementation on a parallel system such as the Department's BBN Butterfly Multi-processor.

Rover's behavior can be characterized as a coping strategy—it accumulates as much information as it can at every moment, but guards against spending an excessive amount of time extracting a particular bit of information and consequently losing track of the objects in the scene. For this reason, the system accumulates information incrementally, preserving as many results as possible if analysis must be cut short during periods of rapid change.

3 Real World Design Constraints

The Rover prototype is constrained by several environmental factors beyond our control. The most important constraint on our design is the computational environment used for the implementation. In particular, the supporting hardware, a Sun-2/120, is a serial machine, with no reasonable facility for exploiting parallelism. With no mechanism for parallel task execution, there is no natural way to realize our task-oriented system organization. At the same time we want our implementation to embody the natural strategy described above. To do this, our system explicitly multiplexes its analysis between the sensory and cognitive levels. The serial nature of the computational resources make it essential to be able to redirect our resources constantly to the most promising task. In a parallel environment, irrelevant tasks do not impair the overall computation as seriously since other computation is proceeding concurrently. In a serial environment irrelevant computation can be disastrous for system performance. To enable processing to focus on only the most promising areas, it is essential that our analysis be broken into small computational tasks. At the end of each task the relative importance of that area of analysis can be re-evaluated.

One of the most telling limitations of the hardware is the bottleneck between the Datacube frame buffer and the Sun. Due to the lack of image processing hardware all image operations have to be performed on the Sun. When the project began, it took a full 8 seconds to transfer the contents of the entire frame buffer to on-board memory. Hand optimizations lowered this figure by a factor of 8, but the significance of the bottleneck is still substantial.

Another major consideration was that the prototype design, once implemented, be easily extensible. We feel there is significant potential for future research involving the Rover system and we want to provide a useful software base for this work. The Rover prototype is designed to facilitate the replacement and addition of functional units. This will ease the eventual replacement of the simple routines of the initial prototype with more sophisticated ones.

4 Prototype Design

The principal goal of the Rover system is to maintain correspondence between multi-frame images of moving objects. To do this many different elements need to be manipulated. The database representing the most current state of the world needs to be maintained. To keep this information up to date, input images must be analyzed to detect areas of motion. These areas of interest in the original image are then correlated with the world database and if necessary have further discrimination techniques applied to them. Since there are potentially multiple areas of interest in any input image but a limited amount of computational power and time to spend processing them, the cycles spent processing each area must be carefully monitored to insure that the information derived from each input image is complete as possible. It is conceivable that any input image will contain more information than the system can process in a reasonable amount of time. If this happens the image and all associated processing is abandoned for a fresh view of the world.

To perform these various tasks, the prototype is broken up into three main modules:

- **Executive**—Responsible for overall system coordination and task scheduling.
- **Raster Segmentation/Motion Detection**—Responsible for detecting areas of blobs in the input image and for segmenting the image into small manageable sub-images.
- **Object Discrimination and Correspondence**—Responsible for identifying the sub-images supplied by the Raster Segmentation module and integrating them into the world database.

Figure 2 diagrams Rover's main functional units and their relations to one another. Briefly, the Executive begins the work on each image frame by enqueueing a batch of "raster scans" tasks. These tasks implement a static search pattern of the current image. While this is going on, the executive watches the clock to avoid spending too much time on any single image frame. (Other strategies could be employed to search the image frame for potential objects—see Appendix C.) Tasks from the Raster Scan Cluster seek light-colored "blobs" in the image frame that may be objects in the scene. The Object Discrimination Cluster

scrutinizes the blobs pointed out by the R.S. Cluster and updates the world database. Thus the Executive performs limited top-down direction of computation, but otherwise, computation proceeds in a bottom-up fashion.

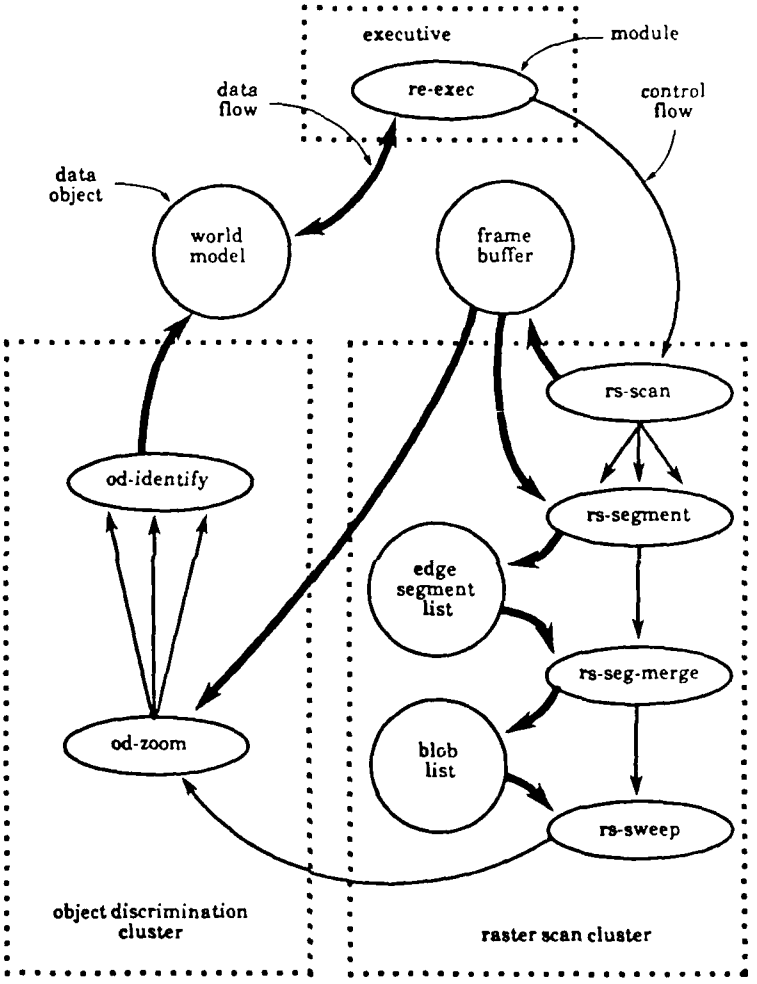


Figure 2: Functional Overview of Rover

Before describing the prototype design in greater detail, we will note the assumptions we made about the world to facilitate the development of the prototype. Because our aim was to develop a system to address a broad range of issues, we constrained the depth of the sensory and cognitive problems Rover would have to solve as follows:

- **Simple Targets**— The targets used for tracking are distinctly-colored spheres. Actual identification of different spheres is done on the basis of past position, velocity,

and "color" (image intensity).

- **Controlled Lighting**— Shadows are not a problem addressed in the initial prototype, so light sources are located behind the camera.
- **Horizontal Motion Only**— Objects are assumed to move in the horizontal direction only. (In fact we were able to loosen this constraint considerably).
- **No Occlusion**— Targets are expected not to occlude one another. Understanding occlusion requires more complicated cognitive and sensory abilities; however, the system achieves limited success in the face of occlusion by assuming the balls participating in the occlusion a single "object."
- **No Complete Replacement**— Balls will not swap positions between successive images.

4.1 Tasks and Inter-module Communication

A *module* is an open-loop[1] process that is invoked by another module. Modules that perform functions related to a particular goal are grouped together in a *cluster*. The cluster is strictly a logical unit used to organize the system: related variable types and *etc.* are shared within a cluster. The actual invocation of a module is done through task creation. The *task* is the actual unit of computation and is created at run-time. It is an instance of a module and performs a portion of the module's work. Tasks are executed according to a priority based on the relative importance of that task to the overall image analysis. Rover implements this lightweight process model by maintaining a priority queue of tasks that are ready to execute. Tasks are enqueued with arguments and once scheduled will run to completion.

In the current implementation, module invocation (beyond the initial enqueuing of enough raster scan tasks for the whole image frame) is driven by the results of each stage of processing. Each module performs its assigned function and enqueues the module whose work should be done next based on the results of the current module. Information is passed between modules either by placing it in a global data structure (as in the raster scan cluster) or by wrapping it up and handing it to the next module as its argument (as in the object discrimination cluster).

A module may be composed of several functions, although it is crucial that each module execute and return quickly so the entire system is not bogged down by a sluggish module. (Rover is intended to be robust enough to adapt to a more rapidly changing environment by processing each frame partially. A module that runs a long time can cause the executive to lose track of the environment because there is no mechanism for interrupting a runaway or hung task.) A module that needs to perform some auxiliary task to continue its computation is thus split into

1. a module to perform the initial computation,

[1] Here *open-loop* means that the process can be dispatched without requiring the invoking module to monitor its progress.

2. a module to accomplish the auxiliary work.

3. and a module to be enqueued by module 2 to conclude the work.

Each module is broken into functional units such that the computation performed by each unit doesn't take too long. At major decision points or functional transitions the next appropriate function is enqueued. [2]

4.2 The Executive

The *Executive* is the framework for all the other functional modules. It coordinates the integration of all the functional units, from the extremely low-level sensory modules that perform pixel operations to the higher level cognitive modules that maintain the world database. The specific functions it performs are controlling task scheduling, task execution, resolving temporally global issues of correspondence and system initialization.

Work is requested in the system by enqueuing tasks on a general work queue. When a task is selected from the queue (because it is the oldest highest-priority task in the queue) it is dequeued and the corresponding code is invoked. The framework (*i.e.*, main loop) looks like the following:

- *Initialize system*
- *Forever*
  - *If the task queue is empty, then grab a new image frame, enqueue a batch of Raster Segmentation tasks for the frame, and reset the interval timer. (See Section 4.2 for a description of the system's notion of time.)*
  - *Get a task from the work queue and invoke it.*
  - *If there is any time left in the current interval, then return to the work queue for more work; otherwise flush the task queue.*

The priorities assigned to tasks are fixed according to the module the task instantiates, but their assignment is not arbitrary. The Raster Segmentation priority is lowest because a raster segmentation task should only be executed when all other processing on other portions of the input image has finished. Once the R.S. module has identified a potential object containing subarea, it will enqueue an Object Discrimination task. There will likely still be R.S. tasks on the work queue, but object discrimination is given priority since there is a high probability of making a positive object identification. This process of object discrimination is also composed of several schedulable steps with the priority of the step being proportional to how high level the task is. Thus, as the system comes closer to identifying an object, it focuses more on that object. Once an interesting sub-image has been processed, lower level tasks still on the work queue will be executed.

[2] Although the modules in each cluster are currently structured as a progression of computational stages, Rover's facilities can support other organizations (*e.g.*, a hierarchical system in which each cluster also has a queue of tasks and one module in each cluster acts as the "executive" for that cluster). See Section 5.1 for more discussion.

It is expected that future work on Rover will incorporate a dynamic priority scheme that takes into account such issues as spatial relevance and confidence levels (see Section 4.4).

Temporal Model of the World

The executive maintains several notions of time. It helps maintain the world database over the course of input images by performing functions not rightly relegated to either of the lower level modules (via the *virtual time stamp*). It also attempts to keep track of the passage of real time in relation to the processing it controls (using the *interval timer*); the executive must prevent excessive computational effort being expended on any one image as the model of the world maintained internally could fall hopelessly out-of-date.

The executive manipulates lower-level modules (*i.e.*, Raster Segmentation and Object Discrimination) to perform the analysis of an image. These routines perform their analyses based on the input snapshot and the past history of the world stored in the world database. An obvious problem with this is that information that is either missing in the current snapshot of the world or that is simply missed by the analysis should still be accounted for in some way. An obvious example would be the movement of an object from the field of view. While it might be reasonable to simply delete the object from the world database, this makes the database extremely volatile; a mistake by the low level routines at any point could result in the accidental but erroneous removal of an object. To counter this problem the system maintains a measure of confidence in each object stored in the world database. Whenever the position and identification of a particular object are reaffirmed, this confidence is raised to the maximum level. Should an image be processed without any new information being provided about an object, for whatever reason, the confidence in its identity (as well as its existence) is lowered. Once this confidence falls below a certain threshold the object is deleted. Hence, if the system fails to process any information pertinent to an object in a long time, it will forget about that object. This provides the system with some measure of resiliency over time.

Another interesting result of this confidence degradation is that the system is capable of dealing with temporary occlusion. In such a case, two objects will enter a spatial relationship causing one of them to be partially occluded. Should they be identified as a single object by the Raster Segmentation and Object Discrimination modules then one of two things will happen. The combined colors of the objects may be identified as a new color and assigned a completely new entry (albeit an erroneous one) in the world database. Alternatively, the object may be identified as one of the existing objects that the system expects to find in the area. In this case there will be a temporary aliasing of one of the objects to the other. Since we assume that the occlusion is temporary, in the first case the objects will separate, the confidence in the erroneous composite "object" will eventually degrade and it will "disappear." In the second case as long as the objects separate soon enough the world model will still contain the aliased object and its position will be updated. In both cases, the temporary loss of correspondence is overcome.

The amount of time spent processing a single image is bounded to prevent falling too far behind the real world. The executive maintains a notion of a virtual time segment, which is the maximum length of time that Rover should spend processing any single input image. If

at the end of a time segment there is still processing to do on the current image, it is likely that any information that could be derived from it would be obtained at the expense of falling behind. To avoid this problem, any tasks that are pending at this point are destroyed and processing on the new image is started by enqueuing new Raster Segmentation tasks. Task destruction is accomplished by simply flushing the queue of any waiting tasks. (This is one reason for tasks being open-loop.) Additionally though, all objects whose internal representation went un-updated have their confidence degraded. Thus the system responds to over-stimulation by ignoring some of the scene to keep up with the world as well as possible.

4.3 Image Segmentation—Locating Objects

The Raster Segmentation (R.S.) cluster isolates the locations of objects in the scene using a coarse sampling of the image. When the executive notices that a potential object has been located, it enqueues an Object Discriminater to examine it. Thus, the Image Segmenter is a cheap filter that saves the expense of transferring uninteresting portions of the image from the frame buffer (a real bottleneck) and allows the system to concentrate expensive high resolution operations on areas of the image that are likely to produce the most useful results.

The Raster Segmenter's design was strongly influenced by the characteristics of the Datacube frame buffer and our interface to it. Grabbing horizontal lines from the frame buffer is faster than any other mode of acquisition, so the R.S. cluster uses a coarse sampling (every 16^{th}) of horizontal "rasters" from the frame buffer to locate potential objects.

Each raster is examined by a line segmenting task. (Each "frame" in the image sequence consists of a pair of images taken in rapid succession from the CCD camera. These images are digitized in the left and the right halves of the frame buffer, respectively.) At the beginning of processing each frame, the executive enqueues a raster segmenting task for each of the rasters that will be scanned in the frame.

Each image segmenter performs these operations on its raster:

1. **Segment the raster**—locate light segments of the raster which might be caused by a light object on the dark field of the background.

2. **Identify pairs of segments**—match pairs of raster segments which appear (and overlap) in both the first and second images; these are *positive* segments (*i.e.*, likely not due to noise in the image).

3. **Identify vertically associated rasters**—grow "regions" of vertically overlapping *positive* raster segments that seem to indicate a single object.

Figure 3 demonstrates the results of segment pairing and object growing. On the left pairs of segments have been identified. A little while later, we see (on the right) that the vertically overlapping pairs have been "grown" into an object region.

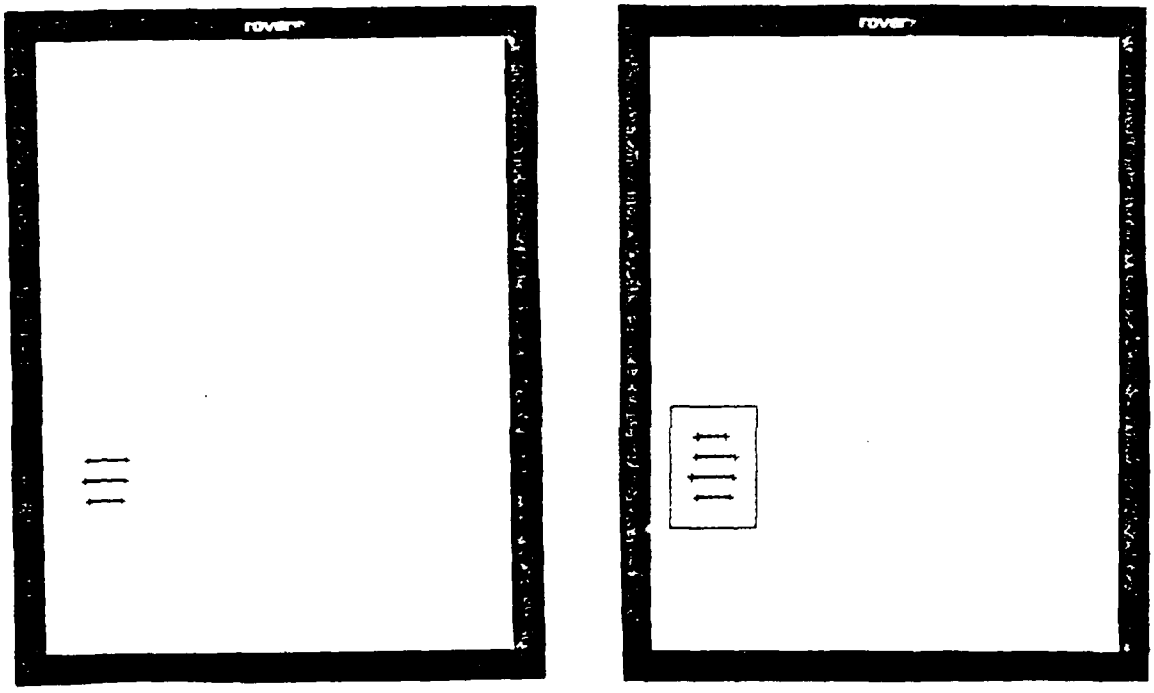


Figure 3: Paired raster segments in the left hand image, and an object region grown from them on the right. Paired segments are denoted by crosses and the object region is enclosed in a box.

Segmenting Rasters

The raster segmenter uses a one dimensional "difference of boxes" edge operator (based on the Kirsch operator) to detect high-contrast edges in its raster from the image. A *positive* segment consists of a *peak-valley pair* in the response of the edge detector. Such a pair should indicate the presence of a light-colored object in the image (against the dark background). The segmenter uses a function of the mean and variance of the intensity of the image raster to determine what response magnitude constitutes a positive response of the edge operator (rather than noise).

Pairing Segments

A pair of segments on a raster (one from each image of the raster) match if the segment in the first image overlaps the segment in the second image. Any unpaired segments are ignored as bogus or noisy false responses. This pairing criterion effectively limits the class of objects the system will recognize to those whose retinal image is not completely displaced in "the blink of an eye."

Thus, a small object traveling at high velocity perpendicular to the line of sight will be ignored because its images in the first and second blinks will not correspond to each other. Conversely, an object of virtually any size speeding directly at the camera will get a very strong match.

Growing Objects from Segments

The R.S. cluster maintains a list of the responses from the image segmenting tasks (one for each raster being scanned). As each segmenting task returns with its list of segment pairs, it updates the record for its raster. In addition, the cluster keeps a list of object regions. As each segmenter returns its list of segments, it tries to update the object list with the new information it is returning about the image. If one of the segments the task found corresponds to one of the objects in the list that segment is added to the object. (A segment corresponds to an object if it is adjacent to the top or bottom of the object, and it overlaps with any of the rasters already in the object, and its velocity—which is estimated from the rapid pair of images—matches closely the velocity of the object.) If the segment matches no current objects, the list of segments is searched for an adjacent (unmatched) segment which could form a new object with the segment being returned. If a segmenter task returns a negative response (no segments found) the object list is searched for objects that are adjacent to the empty raster being returned. Any such objects are bounded by this raster and hence complete. Of course, if a segmenter misses a segment and returns an erroneous negative response, the object region will stop short of the real boundary of the object. An object is also bounded if the adjacent raster contains no segments that match the object (although there may be some segments in the raster). A clean-up task sweeps the object list periodically to detect this condition.

When the clean-up task notices that an object is bounded above and below, it enqueues an object discrimination task to examine the object and try to match it against the current model of the world, as described in Section 4.4.

4.4 Object Discrimination—Maintaining Correspondence

Once blobs have been roughly located by the R.S. cluster, the identification of individual objects must be determined. The initial identification and maintenance of correspondence with particular objects is done by the Object Discrimination (O.D.) cluster. This cluster consists of high level routines used to interface with the executive and low level routines used exclusively by the cluster for processing raw pixel data.

When the R.S. cluster locates an area in which it believes there is an object, a request for closer inspection is put in the system queue and an entry is made to a Temporary Image Buffer (TIB). The queued request will contain a reference to this buffer and will be used when the request is serviced by the Object Discrimination cluster. Note that no part of the image is actually copied out of the frame buffer at this point.

The analysis performed by this module is broken into the following sections:

- **Image Validity**— Insures that the sub-image returned by Raster Segmentation is useful. (See Section 4.4)
  - **Detection of Good Images**—Determines if part of the object in the target image has been cut off by the border of the sub-window.
  - **Detection of Partitionable Images**—Determines if the sub-window contains more than one object in it.
  - **Partitioning of Images**—Partitions the sub-window so that each partition contains only one image.
- **Maintaining the world**—Does the high level correspondence that allows the world database to reflect the state of the observed world at any point in time. This information is used to determine how much low level processing is necessary. (See Section 4.4)
  - **Spatial Matching**—Attempts to identify objects by correlating object position with predictions about the way the world will look given the passage of time and its effect on the world database.
  - **Color Identification**—Expresses the *color* of the object in the input window as the ordered pair containing the mean and variance of the object image intensity.
  - **Color Matching**—Determines if the *color* is one already seen by the system. Assigns each color a unique integer identifier.

Image Validity

In Rover's world, complete identification of an image is a computationally expensive procedure. As a result, it makes sense to attempt to determine whether the information derived from a particular image will be of any use. In this first stage, the dimensions of the window of interest identified by the raster segmenter are passed as parameters. (The raster segmenter avoids reading blocks of the image from the frame buffer as the transfer to Sun memory is an extremely time consuming operation). At this point, the sub-image of interest is transferred from the frame buffer to Sun memory.

The sub-image is checked to determine whether an object straddles any of its edges. This is done by passing a simple threshold operator over each boundary. If a high intensity patch is located, this is taken to indicate that an object is so close to the edge of the window that part of it has been lost. This missing portion may be critical to the correct identification of the object. As a result, we adopt the simple strategy of abandoning further evaluation of this image. However, we return a list of the boundaries of the image on which the object is incident. This information is intended for use in stretching the window to obtain the missing information. This stretching is not implemented in the prototype (see Appendix C).

The sub-image is next analyzed for the presence of multiple objects. This is done by scanning horizontally along the image every few rows. When a high intensity patch is detected, it is considered to be an object. The center of the high intensity patch is calculated and then a line is drawn vertically to determine the upper and lower boundaries of the object. From this information, a more tightly constrained window is drawn around the object. This scanning procedure is continued until the entire input window has been traversed. A list of object dimensions is returned. Figure 4 illustrates the result of this procedure. An object identification task is enqueued for each object found.

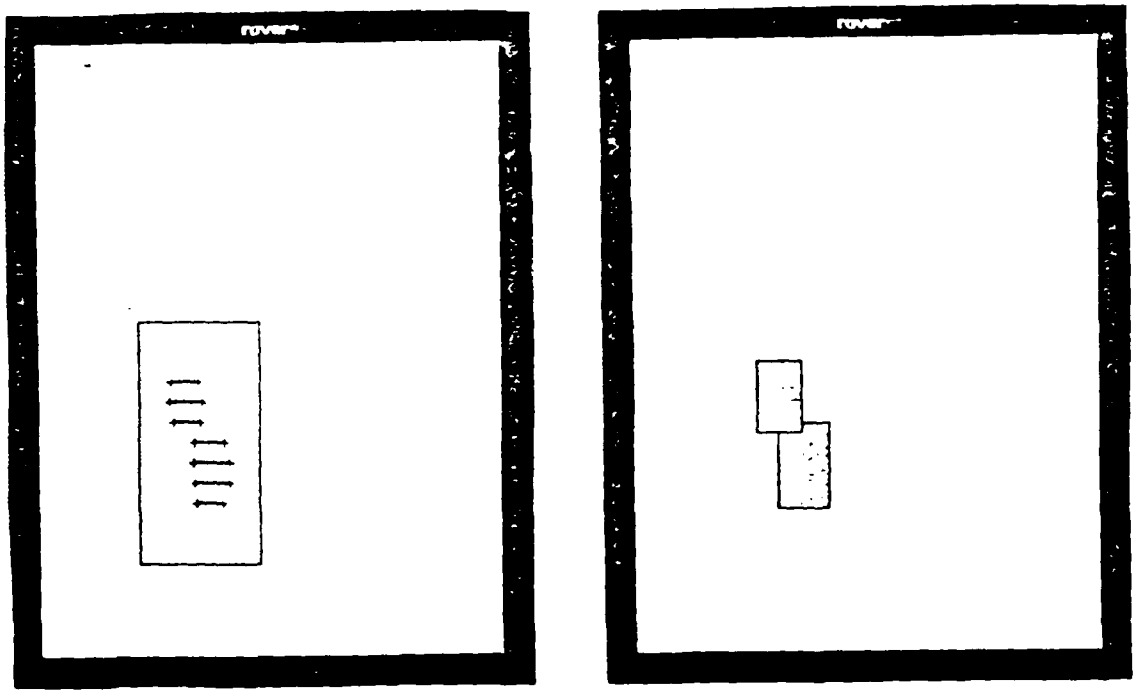


Figure 4: A blob identified by the Raster Segmentation cluster and properly split up by the Object Discrimination cluster.

The techniques used for the object location and window partitioning are admittedly primitive. Edge finding, for example, is done by simple thresholding. The prototype does

use a more sophisticated one dimensional Kirsch operator in the raster segmentation routines (see Section 4.3). The development of the Kirsch occurred concurrent with the development of the object discrimination routines and hence the simple thresholding was chosen to speed the development effort.

Maintaining the World Model

The second phase of the object discrimination process is responsible not only for the accurate identification of objects at a moment in time, but also for maintaining correspondence of objects over time. This phase is where the majority of the maintenance is done on Rover's model of the world. Maintaining correspondence of objects involves doing spatial matching to minimize the search space for correspondence, actually identifying objects by color if necessary, and using the spatial and color information to determine if the object identified is an existing object or a new one. The model is then updated to reflect the results of matching the object to Rover's known world.

The entry of each object in the world database has information about the last determined position of the object and its last known velocity. Using this information it is possible to predict at a given point in time the new position of the object and its approximate velocity. Should the old velocity of an object and the apparent velocity of the object in the window be somewhat different, it may be either that the objects are different or that the object has changed its velocity. To resolve this ambiguity, the expected current position of the object in the database is computed with the observed velocity and this projected position used as a basis for comparison. If the position of the referred window is close to the predicted position of an object then the position of the object in the World is updated and the request is finished. This "dead-reckoning" approach provides a way of conserving computational power, but provides a potentially less accurate picture of the world than could be achieved by always explicitly identifying each object. To compensate for this less accurate correspondence procedure, a confidence measure is associated with each object in the World database. Every time the position of an object is updated without calculating the identity of the object (*i.e.*, with the position correlation described here) the object's measure of confidence is degraded. Once the confidence falls below a certain point the unidentified object will be identified completely.

Once the object is identified either by position or by identity, the appropriate world database entry is updated. The current position of the object and its new velocity are recorded. If the identification is by position then the confidence in that identity is degraded. If the identification is by identity the confidence in the identity of the object is set to 100%. If no entry currently exists in the world database, then a new entry is made.

"Color" is defined in the prototype to be the mean and variance of the intensity in the object. The spheres that serve as our initial targets have the nice property of being relatively invariant in reflectance regardless of the viewing angle. This simplifies identification considerably. To ease the cognitive burden, we assume that all spheres have a distinct color in Rover's world. In order to calculate the mean and variance of the object, its boundaries are located more precisely by using an edge detector. The intensity values within these boundaries are used to calculate the "color" of the object.

Inevitably there are some differences between the mean and variance of two different images of the same-colored sphere. To deal with this problem, we adopt the *ad hoc* approach of considering two "colors" to be the same if they differ by only a small percentage. This percentage is the difference of the weighted sum of the means and variances. If the colors are considered to be the same then the object being identified is assigned an integer identifier associated with that color. Should the difference in the two colors be great enough then the system considers the color to be distinct and assigns it a new identifier.

5 Future Directions

The following section describes work that represents the next logical steps in the full development of the Rover system. It moves Rover closer toward meaningful cognitive interaction with its environment and attempts to solve some of the more basic sensory problems that are unaddressed in the initial prototype. Typical cognitive development involves using the object history and models of object behavior to aid in identification and correspondence. Typical sensory problems are dealing with occlusion and recognizing more complex objects, such as alphabet blocks.

5.1 Architectures and Control Strategies

In the current prototype, the system operates in a bottom-up data-driven fashion with minimal top-down control. This structure is a result of our goal of constucting a system with both sensory and cognitive abilities. Thus the prototype demonstrates (albeit almost trivially) that the architecture supports both bottom-up and top-down control strategies.

Successors of Rover might explore architectures which perform "pre-attentive" perceptual processes bottom-up, without direction from any attentional module. Parallel signal processing is now available to us with our recently acquired MaxVideo (TM) hardware. "Attentional" perceptual and cognitive processes could be controlled by a central module that manages the system's limited computational resources based on the results of the pre-attentive low level processes. The architecture we have developed makes it possible to experiment with various organizations and strategies that could embody such systems.

5.2 Extending Cognition

Future cognitive capacities could use the history of the world to guide the identification process. The executive can make a guess about what the object in a particular window will be and use it to influence which routines in the Object Discrimination module are invoked. The result will be a much more intelligent guess about how to go about processing image windows.

In addition, the executive could use such information to decide which areas of the scene are likely to contain useful information, ignoring other portions during a crucial detail acquisition task.

Conversely, a good clean image of an object could be saved for later processing if the system must devote its attention to keeping up with a scene that is changing extremely rapidly for a short period of time. As long as the old image can reliably be attributed to the correct object by continued tracking, it can be processed at the system's leisure. Note that the system architecture could easily be modified to support this sort of off-line analysis. For instance, such long-term tasks would be retained on the task queue across more than one frame interval, pending an opportunity to indulge in detailed analysis.

5.3 Understanding Occlusion

Section 4.2 describes the prototype's current "approach" to handling occlusion as a consequence of the confidence measure associated with objects in the world database. A more effective technique would be to deal actively with images in which occlusion occurs. In a rudimentary form this would involve explicitly detecting the occlusion and eliminating all further processing of the image. This would prevent the appearance of superfluous and/or erroneous objects in the world database. A more sophisticated solution would attempt to do object identification in spite of the occlusion.

The problem of dealing with occlusion might thus reduced to detecting it. The difficulty of detection increases with the complexity of the objects which Rover is required to recognize. When the cognitive domain is distinctly colored spheres, we could use a segmentation scheme that detects not only binary changes in intensity using edge operators, but intensity changes from one color to another.

When the cognitive domain becomes more complex, say by using alphabet blocks, or using both the blocks and the spheres, the solution might use the straight line detection techniques of [Burns *et al.* 1986] to detect the outlines of the boxes. General areas of interest would be detected using a simple edge operator with region growing. Connected lines could then be grown using the perimeter of the binary region as a starting point. It would then be necessary to define legal relationships between these lines and base the detection decision on an evaluation of these relationships.

An alternative and potentially more successful approach involves simply assuming that there is only one object if multiple objects are not detected. That is, we ignore occlusion. Instead, we attempt to identify the block using the methods described below. One of the constraints mentioned below is that the figures (letters) that may appear on the blocks are cataloged beforehand, giving Rover a source against which to compare potential identifications. If the regions in the object can be matched to one of these known id's then the information is used. The gamble is that no information (or worse, erroneous information) will be derived if the images are occluding. Empirical results will be needed to determine whether occlusion will make identification so difficult as to justify the expense (both in computation and development) of good techniques for detecting the occlusion so as to prevent the useless expenditure of computation time.

5.4 Recognizing Alphabet Blocks

A natural extension of Rover's current capabilities would be to introduce more structure into the visual environment. This would put more emphasis on fast image analysis without modifying Rover's basic control strategy. The new domain would encourage incorporation of new pipelined video-rate image processing hardware in our vision laboratory.

Identification and tracking of alphabet blocks is significantly more complex than the corresponding identification problem with the multi-colored spheres. Issues of rotation, projective distortion, and character recognition must all be resolved. The solutions employed must be fast and effective. They need not, however, work all the time. As long as characters can be effectively recognized most of the time when given a good view, enough information can be gathered to maintain correspondence.

The proposed solution to this recognition problem is strongly reminiscent of Constructive Solid Geometry Techniques [Ballard and Brown 1982] and is composed of:

- Edge Detection
- Segmentation/Blob Growing
- Blob Relations

The edge detection facilitates the segmentation. The segments are then partitioned into faces and the region described by each face is compared to a dictionary of face relations that describe, in a slant-invariant and rotation-invariant form, the various characters that are known to the system.

To do segmentation we propose a linear region growing approach similar to the one used by the current prototype to grow raster segments into "objects" (see Section 4.3). This allows us to segment the image into blobs in time that is almost linearly proportional to the size of the image. A collapsing union-find technique such as that used by [Bukys 1986] can achieve the needed speed. The segmentation parameterizes the blobs by size and center of mass. We may also have special-purpose hardware capable of this segmentation in the near future.

Once the image has been segmented into blobs, the blob identifiers are inspected and fully related to one another. This process yields relationships such as which blobs surround which, how many blobs are contained in a particular blob and how many blobs are adjacent to a particular blob. This information should provide a way of quickly discriminating between different characters as long as the blob relationships describing them are significantly different. An obvious example is the difference between X's and O's. The X has a one region containing another, the O a region containing a region that contains yet another region. The amount of information stored is small and is easily compared against a dictionary of known recognizable figures.

This technique, which is similar to the relational matching of [Barrow and Popplestone 1971], appears preferable to moments [Alt 1962] because it is considerably faster and more stable. ([Alt 1962] advises the calculation of several moments to ensure accurate character identification.) The blob technique provides a means for identifying most distinct figures (it

would doubtless have trouble distinguishing between "I" and "l") that is completely invariant with regard to rotation about the z-axis. We would also argue that it is more invariant with regard to the skewing that results when a cube rotates about its vertical axis.

Of course, due to rotation about the y-axis (vertical), there will be times when the view of the target block is such that the images of the characters on the sides of the target are distorted beyond recognition. Moreover, if the image is bad enough, it may not be worth the computational effort to extract the identification using any technique. In this case the approach which seems best is to simply relax until a better view is available.

The effectiveness of this algorithm hinges on the ability to segment images consistently. If the segmenter produces different blobs and blob relations when the image is perturbed at all then the technique will fail miserably. We can control the experimental environment somewhat to ease the burden on the segmentation routine, but only experimentation with real images will indicate whether this is a fruitful approach.

6 Conclusion

Rover is a working system that exhibits both sensory and cognitive abilities with real world images. The architecture we devised in the course of designing the prototype provides a stable base from which to launch further experimentation with active vision systems in the laboratory. The system is easily extendible, and the structures can support a wide variety of organizations. Future work can incorporate real-time hardware, both for image analysis and sensor control, into Rover.

References

[Alt 1962] F. L. Alt, "Digital Pattern Recognition by Moments", *Journal of the ACM*, 9:240–258, 1962.

[Ballard 1987] Dana H. Ballard, "Eye Movements and Visual Cognition", Technical Report 218, University of Rochester, Computer Science Department, June 1987.

[Ballard and Brown 1982] Dana H. Ballard and Christopher M. Brown, *Computer Vision*, Prentice-Hall, Inc., 1982.

[Ballard *et al.* 1987] Dana H. Ballard, Christopher M. Brown, David J. Coombs and Brian D. Marsh, "Eye Movements and Computer Vision", *1987-88 Computer Science and Engineering Research Review*, University of Rochester, Computer Science Department, Rochester, NY, September 1987.

[Barrow and Popplestone 1971] H. G. Barrow and R. J. Popplestone, "Relational Descriptions in Picture Processing", *Machine Intelligence 6*, Edinburgh University Press, Edinburgh, 1971.

[Bukys 1986] Liudvikas Bukys, "Connected Component Labeling and Border Following on the BBN Butterfly Parallel Processor", Butterfly Project Report 11, University of Rochester, Computer Science Department, October 1986.

[Burns *et al.* 1986] J. B. Burns, A. R. Hanson and E. M. Riseman, "Extracting Straight Lines", *IEEE Transactions on Pattern Analysis and Machine Intelligence*, 8:425–455, 1986.

[Kernighan and Plauger 1978] Brian W. Kernighan and P. J. Plauger, *The Elements of Programming Style*, McGraw-Hill Book Company, second edition, 1978.

[LeBlanc *et al.* 1986] T. J. LeBlanc, N. M. Gafter and T. Ohkami, "SMP: A Message-based Programming Environment for the BBN Butterfly", Butterfly Project Report 8, University of Rochester, Computer Science Department, July 1986.

[Scott 1986a] M. L. Scott, "The Interface between Distributed Operating System and High-level Programming Language", *Proc. International Conf. on Parallel Processing*, pages 242–249, August 1986.

[Scott 1986b] M. L. Scott, "LYNX Reference Manual", Butterfly Project Report 7, University of Rochester, Computer Science Department, March 1986.

A Current Implementation

This appendix briefly describes various parts of the prototype to provide ties to discussion of Section 4 which is divorced from the code for clarity. Figure 5 depicts the topology of Rover's source code files. This should serve as a guide to the source code. The clusters and libraries are briefly described in this section to provide an introduction to the code in the current implementation.

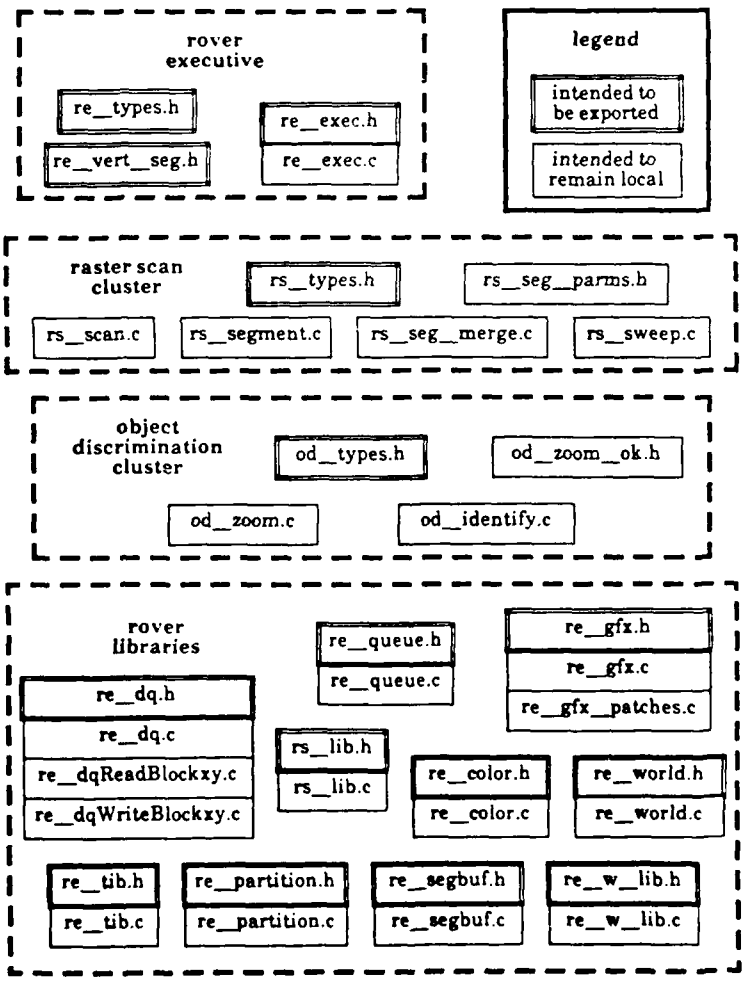


Figure 5: Rover's Source Code Files

A.1 Executive (re_exec)

Rover begins execution in the Executive, setting the command line options and initializing the global data structures. The Executive then starts the first frame interval.

At the beginning of work on each frame, the Executive notes the time and enqueues enough raster scans to search the entire frame image at a predefined density (every 16 horizontal lines). When all interesting work has completed or time runs out on the current frame (the time limit is a compile-time system parameter) the Executive flushes the queue and degrades the confidence[3] of each object in the world database (DB). The confidence of an object is degraded the most if it was not updated at all during the previous frame interval. The confidence is degraded a little if its position and motion were updated based only on location and size correspondence with the image, and it not degraded at all if the "color" of the item was used in the correspondence check (*i.e.*, the object's identity was "completely" verified, assuming each object has a unique "color").

A.2 Clusters

Raster Scan Cluster (rs_*)

This cluster scans the image frame coarsely to quickly identify blobs in the image and estimates the horizontal motion of each blob found. (Recall that the objects in the scene are assumed to move in horizontal planes.)

Rs_scan is called by the Executive to enqueue the scans initially; it also takes a new image in the frame buffer. (Actually, each image frame consists of a pair of images at one-half horizontal resolution. Rs_segment uses these two images to estimate the horizontal velocity of each detected segment. Subsequent modules use only the second image of the pair.)

Rs_segment recognizes a segment[4] in the pair of images if the segment appears in both frames and its images overlap spatially (if the two images are overlaid on one another). Each detected segment's horizontal velocity is also estimated, and all the segments detected on this horizontal raster line are recorded in the segment list. Then an rs_seg_merge is enqueued to work on these results.

Rs_seg_merge tries to merge each segment found on the indicated raster line either into an existing blob[5] or with another unmerged segment to create a new blob. A segment may merge into a blob or with another segment if they overlap and their estimated velocities are "close" to one another (as determined by a pre-defined percentage-error threshold). Then an instance of rs_sweep is enqueued.

Rs_sweep sweeps through the blob list to cap off existing blobs by noting a lack of segments in the expected positions on the rasters above and below the known extent of each blob. For each blob that is apparently bounded by background, an od_zoom is enqueued to examine that region of the image.

---

[3] The *confidence* of an object reflects the quality of the data about the object.

[4] A *segment* is a bright section of a horizontal line that is surrounded dark sections

[5] A *blob* is a set of vertically adjacent and horizontally overlapping segments whose velocity estimates are compatible

Object Discrimination Cluster (od_*)

This cluster examines at finer resolution each subimage identified by the Raster Scan Cluster as a potential object and uses its results to update the world database.

Od_zoom copies the indicated subimage from the frame buffer into a temporary image buffer. If an object appears to be "clipped" by any edge of the subimage (*i.e.*, the object is not completely contained in the subimage) the subimage is discarded; otherwise, an od_identify is enqueued for each object found in the image.

Od_identify spatially back-projects its object-image onto the world DB to find objects already known in the world to which this image might correspond. If no objects of about the same size are found in the expected locations, or the confidence of the closest match is low (*i.e.*, the accuracy of the information on this object is suspect) the image's "color" (mean and variance of brightness) is calculated and the world DB updated by the best match (if any are "close") in color, size, and *etc.*. If no existing object matches the image well enough, a new object is placed in the world DB.

A.3 Libraries

Task Queue Manager (re_queue)

This library provides the operations *new_queue*, *enqueue*, *dequeue_highest*, *dequeue*, *and q_flush* operations on instances of priority queues of tasks. A Module is enqueued as a pointer to a function, with a pointer to a structure that holds the function's arguments, a function to free the argument in case of a q_flush, and the module's priority.

Graphics Display (re_gfx)

The graphics display runs under SunTools. It provides the ability to start up Rover's display window, clear it, and draw crosses, lines and boxes in the window. These facilities are used by the clusters to display their results as Rover runs.

Datacube Interface (re_dq)

This library implements functions to digitize a new image frame in the frame buffer, and to get any subwindow of the frame buffer for closer inspection.

Binary Line Segmenting (rs_lib)

A one-dimensional Kirsch edge detector and general-purpose line segmenter are implemented. The edge detector is a simple "difference of boxes" applied to each point on the line (image vector) that is an argument to the edge detector. Thus the result of the edge detector is a magnitude vector in which rising edges of brightness in the image vector appear as peaks, falling edges appear as valleys, and segments of constant brightness give no response. The edge detector also returns the mean and standard deviation of the

brightnesses of the points on the line. The line segmenter locates "peak-valley" pairs whose absolute values exceed a threshold supplied to the function as an argument.

Segment and Blob Lists (re_segbuf)

Essentially, the operations new, insert, and member (like the operations on the abstract data type, set) are implemented for use on segment and blob lists.

Temporary Image Buffers (re_tib)

The operations new, get_image, and free are provided for using the Temporary Image Buffers (TIBs). TIBs are used in the Object Discrimination Cluster to hold subimages from the image frame.

Image Partitioning (re_partition)

Facilities are provided for searching within and splitting image partitions in TIBs.

Object Color Identification (re_color)

This library implements functions to calculate the "color" of an image, and to match a color against the system's current registry of colors. The color an object is the pair (mean, variance) of the brightness of the image of the object. It is assumed to uniquely identify an object.

World Database (DB) Manager (re_world)

This library provides the interface to the spatially indexed world model. It implements the operations necessary initialize the DB, search a spatial region of the world, attempt to match a given object with an existing one in the DB, update an existing object with new information, insert a new object in the DB, and degrade the information in the DB.

B Building on Rover

For the reader who wishes to extend the current implementation of Rover to realize greater functionality, this section is devoted to outlining our mistakes for your benefit, and indicating obvious directions for extending Rover.

B.1 Hints for Anguish-free Hacking

We offer these suggestions for your enhanced hacking pleasure (take them with as many grains of salt as you like):

1. Read *The Elements of Programming Style* [Kernighan and Plauger 1978]. At least scan the Summary of Rules at the end—it only takes two minutes, and the reminders may save you many horrors we were not spared in the initial implementation.

2. Follow the conventions established in the existing code. Each lends itself to clear code and minimum interaction between source code in separate files.

3. Adopt the following convention regarding allocating and freeing memory for temporary results:

   - Any library that exports a function which returns `malloc'd` memory as a result must also export a function to *free* such objects as its functions create.
   - Any function using a function that returns `malloc'd` memory is responsible for utilizing the associated *freeing* mechanism to prevent memory leaks.

4. Of course, it is even more important to avoid `free'ing` memory that other functions may reference in future, as this leads to unpredictable results.

5. Try to use a tight design-implement-test cycle to keep changes as incremental as possible. And of course keep backup copies of the latest version of working code. (RCS is fairly nice for this.)

C Extensions

There are several clear directions that future work with Rover could take. We list some here:

- Do something better with clipped windows than discarding them. For instance, try getting a subimage from the frame buffer adjacent to the clipped edge to grow the window in hopes of finding the whole object. Windows are clipped frequently in practice, so this could lead to significant gains in performance. It could, of course, be argued that the Raster Scanner should be improved to reduce this frequency. Also, in the presence of occluded images (consider several balls, each overlapping) we might not be able to afford to discard partial images.

- Replace simple thresholding in the Object Discrimination Cluster with sparse application of an edge operator to locate approximate boundaries of objects. (The rs_lib was not available when the O.D. cluster was implemented.)

- Extend the declaration of the return values of modules to be a `struct union` and use this to explore top-down strategies for directing the search of the image frame and processing of located blobs.

- Experiment with the compile-time parameters (*e.g.*, confidence thresholds, error measures and limits) of the system to improve performance in any or all modules and libraries.

- Explore other structures for organizing the system (*e.g.*, hierarchical—see Section 5.1) and other control strategies (*e.g.*, more direction by the executive, dynamic search for blobs, and using dynamically assigned priorities to direct processing).

- Extend the system to recognize and handle other types of objects (*e.g.*, cubes) more sophisticated identification methods (*e.g.*, markings on objects) and to understand occlusion explicitly.

- Extend the model and system to include camera movements and to support some of the image analysis presented earlier.

- Extend the model to two cameras.

- Extend the system to exploit our MaxVideo (TM) pipeline processor for low level operations.

- Harness the MIMD power of our Butterfly (TM) multiprocessor to do *real* parallel processing using SMP (Structured Message Passing) process families [LeBlanc *et al.* 1986] or LYNX [Scott 1986b,Scott 1986a].

C.1 Templates for Your Own Code

To illustrate the basic forms of the main types of Rover's components, Figure 6 gives a sample of how the skeleton of a cluster declaration file should look. Similarly, Figure 7 describes how a module in that cluster might appear, and Figures 8 and 9 illustrate the essential structure of a library.

D Rover's Code

We conclude with more detail on the existing source code itself. The code can be found on the system in `/u/coombs/projects/rover`. An executable is available to be run (on the Sun with the Datacube—currently betelgeuse) and is located in directory `/u/coombs/projects/rover/bin`. It is invoked from the shell by

```
rover [-c<camera#>] [-f<follow-target#>]
```

Rover expects to run in the Sun View environment, and its graphics interface appears in the upper left quadrant of the screen.

D.1 Coding Conventions

Several conventions are followed in the Rover code. Some of the more helpful ones are listed here.

```
  /* Rover Sample Cluster Declaration Template --- for EXPORTING
     declarations to modules that need to know your types, etc to
     interact with this cluster.   */

#ifndef CLUSTER_TYPES                /* Protect your declaration files from
                                     being included more than once.
                                     Hence, this #define'd constant must
                                     be unique in your system.  That's
                                     why we base it on the file name, as
                                     a convention.  */
#define CLUSTER_TYPES 1

  /* Include only those declarations needed to declare the types, etc
     that you declare here.   */
#include "re_types.h"                /* in case you need global declarations */
#include "re_queue.h"                /* needed to declare your modules */



  /* Module parameter types are declared here so other modules can
     construct arguments for your modules.   */
typedef struct {
  int arg1;
  float arg2;
} mod_parm_t;


/* export modules in this cluster so other modules can enqueue them on
   the task queue.   */
extern q_func_t this_module();

#endif CLUSTER_TYPES
```

Figure 6: Sample Cluster Declaration (cluster_types.h)

```
  /* Rover Sample Module Template --- Include declarations of global
     system, libraries, and other clusters that this module interacts
     with.   */

#include "re_types.h"                /* Always include rover's global types */
#include "re_exec.h"                /* declaration of global data objects */

#include "re_queue.h"                /* any necessary libraries */
#include "re_gfx.h"

#include "other_cluster_types.h" /* a cluster this module will
                                    interact with */

#include "cluster_types.h"        /* this cluster's declarations */



q_func_t                         /* every module returns type q_func_t
                                    (declared in re_queue.h) */
module(my_parm)
     mod_parm_t * my_parm;
{
  other_mod_parm_t * other_mod_args; /* declared in other_cluster_types.h */
  q_func_t return_value;


  /* my processing here */

  /* enqueue another module on the global task queue (work_q, from
     re_exec.h) to perform the next logical operation based on what I
     have seen.  Coerce the type of the args-ptr to what the queue
     library expects.  */
  enqueue(work_q, other_module, (q_arg_ptr_t) other_mod_args,
          arg_free_fn, OTHER_MOD_PRIO);

  return (q_func_t) return_value; /* return a value */
}
```

Figure 7: Sample Module Source (module.c)

```
  /* Rover Sample Library Declaration Template --- for EXPORTING
     declarations to modules and libraries that want to use the
     facilities you provide.    */

#ifndef LIBRARY                          /* Always protect your declaration
                                        files from be included more
                                        than once. */
#define LIBRARY 1

#include "re_types.h"                    /* any declarations needed */



  /* Library function parameter type declarations */
typedef some_type arg_type1;
typedef global_type arg_type2;           /* global_type declared
                                        in re_types.h */
typedef return_type func1_t;
typedef another_type func2_t;


/* export functions in this library */
extern func1_t func1();
extern func2_t func2();

#endif LIBRARY
```

Figure 8: Sample Library Declaration (library.h)

```
  /* Rover Sample Library Template --- this file (library.c) contains
     the library functions advertised in library.h and any internal
     utilities that are expected to be of use only in implementing
     this library's facilities.   */

#include "re_types.h"              /* Always include rover's global types */
#include "re_exec.h"               /* declaration of global data objects
                                   if necessary */

#include "re_queue.h"              /* any necessary other libraries */
#include "re_gfx.h"

#include "library.h"               /* my declarations */



func1_t                              /* lib functions declare their
                                   return types */
func1(arg1, arg2)
     arg_type1 arg1, arg2;
{
  func1_t return_value;

  /* my code here */

  return (func1_t) return_value; /* return my value */
}



func2_t                              /* lib functions declare their
                                   return types */
func2(arg1, arg2)
     arg_type2 arg1, arg2;
{
  func2_t return_value;

  /* my code here */

  return (func2_t) return_value; /* return my value */
}
```

Figure 9: Sample Library Source (library.c)

- **Declaration files** are protected against being multiply included (leading to redefinitions of objects and types) by defining a constant upon the first inclusion that acts as a *guard* against subsequent inclusion during a single compilation. (See Figures 6 and 8.)

- **Type definitions** are named in either of two common forms:

  1. `NEW_TYPE` in all capital letters, or
  2. `new_type_t` in lower case, with the suffix "`_t`."

- **Declarations** are made as local as possible to avoid interference among types, data objects, and functions.

- **File name prefixes** refer to the major component of the system to which the file belongs. The three primary prefixes in the current code are:

  1. `re_`—the system at large (Roving Eyes)
  2. `rs_`—the Raster Scan cluster
  3. `od_`—the Object Discrimination cluster.

END

DATE

FILMED

MARCH

1988

DTIC